National Institute of Technical Teachers' Training & Research
Block - FC Sector - III Salt Lake City
Kolkata - 700 106

Self Instructional Notes for Learning Elementary Commands of **UNIX**

LOGIN: Once the terminal is power on, the following message appears on the screen.

Login:

Every user has an identification name called the user name or login name, which he/she has to enter at the login prompt.

After entering the login name, prompt for password appears on the screen.

Password:

The user is now supposed to enter his/her password.

In response to a valid **user name** followed by a valid **password**, the system display the **$** symbol on the screen. Any valid **UNIX** command can be entered at this stage.

To change password

```
$passwd
Old Password:
New Password:
Reenter Password:
$
```

When system will prompt for Old Password, user has to key in at this prompt, it then prompt for new password twice, user has to key in according to password naming rules applicable to the system. As expected, the characters that are key in are not echoed to the terminal

Logout: Press **^d** at **$** prompt.

COMMANDS

To display todays date and current time

```
$date
Wed Feb 02 11:43:34 IST 1994
$
```

To display device name of the terminal where user is now working

```
$tty
/dev/ttti15
$
```

To display the calender of the month April 1993

```
$cal 4 1993
April 1993

S       M       Tu      W       Th      F       S
                                1       2       3
4       5       6       7       8       9       10
11      12      13      14      15      16      17
18      19      20      21      22      23      24
25      26      27      28      29      30
$
```

To display every one who is currently logged in

```
$who
general         tty01   Feb 02 12:35
subrata         ttyi14  Feb 02 12:42
ranjan  ttyi15  Feb 02 12:41
$
```

To display user name and corresponding information

```
$who am i
general         tty01   Feb 02 12:35
$
```

To display a text on the screen

```
$echo "Welcome to Unix system"
Welcome to Unix system
$
```

To create a file

```
$cat > abc
Welcome to Unix system
^d
$
```

*** File name should be within 14 characters ***

To type (concatenate) a named file on the screen

```
$cat abc
Welcome to Unix system
```

```
$
```

To create a new file

```
$cat > xyz
Welcome Unix
MultiUser System
^d
$
```

To display two files with a single command

```
$cat abc xyz
Welcome to Unix system
Welcome Unix
MultiUser System
$
```

To list file names

```
$ls
abc
xyz
$
```

To list file names in reverse order

```
$ls -r
xyz
abc
$
```

To List files sorted by time modified (latest first)

```
$ls -t
xyz
abc
$
```

To List all files including . files

```
$ls -a
.
..
.lastlogin
.profile
abc
xyz
$
```

To long list files

```
$ls -l          User         Size              File Name
total 4          |                   |                  |
-rw-r--r--     1 general group       23 Feb 02 12:48 abc
-rw-r--r--     1 general group       18 Feb 02 12:49 xyz
$              |        |              |
```

No. of links   Group   Date & time of last modification

the file type as the *first character* on the extreme left

'**-**' ordinary files
'**d**' directory files
'**b**', '**c**', etc. different types of special files

next 9 characters indicates **F**ile **A**ccess **P**ermission (FAP)
- first 3 character indicates **FAP** for file owner
- next 3 character indicates **FAP** for group owner
- last 3 character indicates **FAP** for others

*** **r**ead, **w**rite & E**x**ecute ***

To combine two options of ls command

```
$ls -tl
total 4
-rw-r--r--      1 general group       18 Feb 02 12:49 xyz
-rw-r--r--      1 general group       23 Feb 02 12:48 abc
$
```

To change file access rights (change mode)

*To Remove **owners** write permission for **abc** file*

```
$chmod u-w abc
$ls -l
total 4
-r--r--r-- 1 general group   23 Feb 02 12:48 abc
-rw-r--r-- 1 general group   18 Feb 02 12:49 xyz
$
```

*To add **groups** write and execute permission for file **abc***

```
$chmod g+wx abc
$ls -l
total 4
-r--rwxr-- 1 general group   23 Feb 02 12:48 abc
-rw-r--r-- 1 general group   18 Feb 02 12:49 xyz
$
```

*To remove **others** read permission for **abc** file*

```
$chmod o-r abc
$ls -l
total 4
-r--rwx---      1 general group       23 Feb 02 12:48 abc
-rw-r--r--      1 general group       18 Feb 02 12:49 xyz
$
```

*User can use octal number for **FAP** like follows*

```
$chmod 745 abc
$ls -l
total 4
-rwxr--r-x          1 general group      23 Feb 02 12:48 abc
-rw-r--r--    1 general group      18 Feb 02 12:49 xyz
$
```

** **all** permission for **file owner** **
** **r**ead permission for **group** **
** **r**ead & **w**rite permission for **others** **

** *Octal*	*Binary*	*FAP* **
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

To Count no. of lines, words and characters in a file

```
$wc abc
1      4      23 abc
$
```

No. of characters

```
$wc -c abc
23 abc
$
```

No. of words

```
$wc -w abc
4 abc
$
```

No. of lines

```
$wc -l abc
      1 abc
$
```

No. of words and lines

```
$wc -lw abc
      1      4 abc
$
```

To Copy content of one file to another

```
$cp abc arc
$cp xyz byz
$ls
abc
arc
byz
xyz
$
```

To Change file name

```
$mv byz crp
$ls
abc
arc
crp
xyz
$
```

To Remove a file

```
$rm crp
$ls
abc
arc
xyz
$
```

To Remove a file after confirmation

```
$rm -i arc
arc: ? n
$ls
abc
arc
xyz
$
```

To display specified no. of lines from a file

To explore various options under this feature of **UNIX**, create a file **file5** containing all lower case alphabet

```
$cat > file5
a
.
.
z
^d
$
```

To display first 10 lines of the file

```
$head file5
a
b
c
d
e
f
g
h
i
j
$
```

To display first 5 lines of the file

```
$head -5 file5
a
b
c
d
e
$
```

To display last 10 lines of the file

```
$tail file5
q
.
.
z
$
```

To display file starting at line 5

```
$tail +5 file5
e
f
.
.
z
$
```

To display last 5 lines of the file

```
$tail -5 file5
v
.
.
z
$
```

To printing working directory

```
$pwd
/usr/general
$
```

To create a sub directory

```
$mkdir dir1
$ls -l
total 6
-rw-r--r--    1 general group    23 Feb 02 12:48 abc
-rw-r--r--    1 general group    23 Feb 02 14:48 arc
drwxr-xr--    2 general group    32 Feb 02 15:28 dir1
-rw-r--r--    1 general group    18 Feb 02 12:49 xyz
$
```

To Change current directory

```
$cd dir1
$pwd
/usr/general/dir1
$
```

To copy abc file from /usr/general to current directory

```
$cp /usr/general/abc .
$ls
abc
$
```

To change login directory to current directory

```
$cd
$pwd
/usr/general
$
```

To remove a sub directory

```
$rmdir dir1
rmdir: dir1: Directory not empty
$
```

The following sequence of commands takes the user to the sub directory *dir1*, removes all the files from that sub directory and finally takes the user back to the *login directory*.

```
$cd dir1
$rm *
$ls
$cd
$pwd
/usr/general
$rmdir dir1
```

```
$ls -l
total 5
-rw-r--r--     1 general group       23 Feb 02 12:48 abc
-rw-r--r--     1 general group       23 Feb 02 14:48 arc
-rw-r--r--     1 general group       18 Feb 02 12:49 xyz
$
```

To display space consumed by the current directory as well as its sudirectories (in blocks, block=512 bytes)

```
$du
48      ./bs
22      ./awk/awwk
150     ./awk
354     .
$
```

To display space consumed by the each file of the current directory as well as its subdirectories

```
$du -a
2       ./.profile
2       ./file1
2       ./file2
2       ./bs/b1
2       ./bs/b2
48      ./bs
2       ./awk/country
10      ./awk/t
150     ./awk
2       ./a
2       ./b
354     .
$
```

Comparing files

To explore various options under this feature of **UNIX**, create following files

```
$cat > file1
abcd
def
ghi
^d
$
$cat > file2
abce
def
ghh
^d
$
```

*The **cmp** command*

This command compare two files and display the first line number and character number at which the two files differ

```
$cmp file1 file2
file1 file2 differ: char 4, lin 1
$
```

*The **comm** command*

This command compare two files and list lines that are uniq to each files and lines that are common to both file

```
$comm file1 file2
abcd
        abce
                def
        ghh
ghi
$
```

first column - lines *only in the file1*
second column - lines *only in the file2*
third column - lines *common to both files*

To display only common lines

```
$comm -12 file1 file2
def
$
```

To supress common lines

```
$comm -3 file1 file2
abcd
        abce
        ghh
ghi
$
```

To supress uniqe lines from file1

```
$comm -1 file1 file2
abce
        def
ghh
$
```

Locating files using *find* command

find *<path list> <selection criteria> <action>*

*To display all files in all the sub-directories begining from current directory (**.**)*

```
$find . -print
.
./.profile
./.lastlogin
./file1
./file2
./bs
./bs/b1
./bs/b2
./awk
./awk/country1
./awk/t
./awk/awk
./a
./b
$
```

To display only directories in all directories begining from current directory

```
$find . -type d -print
.
./bs
./awk
$
```

To display only ordinary files in all directories begining from current directory

```
$find . -type f -print
./.profile
./.lastlogin
./file1
./bs/b2
./awk/country
./awk/t
./a
./b
$
```

*To display specified file named **b1** in all directories begining from current directory*

```
$find . -name b1 -print
.b1
./bs/b1
$
```

*To display specific files using wild-card characters (wild-card character must be specified in **double quotes**.)*

```
$find . -name "file?" -print
./file1
./file2
$
```

```
$find awk -name "a?" -print
awk/a1
awk/a2
awk/awwk/a1
awk/awwk/a2
$

$find . -name "*.c" -print
./a.c
./awk/b.c
$
```

*To perform file operations on the files that are located by the **find** command.*

```
$find awk -name "a?" -print -exec cat {} \;

awk/a1

BEGIN {OFS=":"}
 {print $1,$2,$3,$4}

awk/a2

BEGIN {w=0;c=0}
 {w=w+NF
  c=c+length($0)+1}
END {printf " %6d %6d %6d %s\n",NR,w,c,ARGV[1]}
$
```

*To perform file operations on the files that are located by the **find** command after **confirmation***

```
$find awk -name "a?" -ok cat {} \;

<cat ... awk/a1 > ? y

BEGIN {OFS=":"}
 {print $1,$2,$3,$4}


<cat ... awk/a2 > ? n
$
```

To display files that are created or updated on 15 days before the current date

```
$find . -mtime 15 -print
./.profile
./bs/b1
$
```

To display files that are created or updated earlier than 15 days before the current date

```
$find . -mtime +15 -print
./file1
./file2
$
```

To display files that are created or updated in the last 15 days before the current date

```
$find . -mtime -15 -print
.
./.lastlogin
./bs
./bs/t
./awk
$
```

Sorting a file

To explore various options under this feature of UNIX, create following files

```
$cat > file1
ps
kill
ls
345
Ln
cut
238
^d
$
```

```
$cat > file2
1
234
88
100
245
888
^d
$
```

```
$cat > file3
235:982:187
187:467:235
982:187:467
467:235:982
^d
$
```

To sort file ***file1***

```
$sort file1
238
```

```
345
Ln
cut
kill
ls
ps
$
```

*To sort file **file1** ignoring case*

```
$sort -f file1
238
345
cut
kill
Ln
ls
ps
$
```

*To sort file **file1** in reverse order*

```
$sort -r file1
ps
ls
kill
cut
Ln
345
238
$
```

*To combine two options of **sort** command*

```
$sort -rf file1
ps
ls
Ln
kill
cut
345
238
$
```

*To sort a numeric file **file2***

```
$sort file2
1
100
234
```

```
245
88
888
$
```

This is not a currect solution. Sort command generally compare two lines by ASCII value. To perform numerical comparison

```
$sort -n file2
1
88
100
234
245
888
$
```

See the file ***file3****. It has three fields and they are separated by a :. The* ***:*** *is called* ***field delimeter****. To sort the file file3 on the basis of 2nd field*

```
$sort -t":" +1 -2 file3
982:187:467
467:235:982
187:467:235
235:982:187
$
```

To store the result of sort command in to a file

```
$sort -t":" +1 -2 file3 -o file4
$cat file4
982:187:467
467:235:982
187:467:235
235:982:187
$
```

Searching a pattern within a file

To explore various options under this feature of **UNIX** create a file as shown below

```
$cat > def
Use the grep command to select
and extract lines from a file,
and print only those lines
that match a given pattern.
Enter the following command to
print out the lines in 'def'
that contain "to". There are
only three such lines.
^d
$
```

*To search lines containing "**to**"*

```
$grep "to" def
Use the grep command to select
Enter the following command to
that contain "to". There are
$
```

To display the line no. followed by the matching line

```
$grep -n "to" def
1:Use the grep command to select
5:Enter the following command to
7:that contain "to". There are
$
```

To display the lines not matching with the given pattern

```
$grep -v "to" def
and extract lines from a file,
and print only those lines
that match a given pattern.
print out the lines in 'def'
only three such lines.
$
```

To display total no. of lines that matched with the given pattern

```
$grep -c "to" def
3
$
```

*To combine the different options of the **grep** command*

```
$grep -cv "to" def
5
$
```

*To display all lines that are starting with '**t**'*

```
$grep "^t" def
that match a given pattern.
that contain "to". There are
$
```

*To display all lines having '**th**' followed by any character*

```
$grep "th." def
Use the grep command to select
and print only those lines
that match a given pattern.
Enter the following command to
```

```
print out the lines in 'def'
that contain "to". There are
only three such lines.
$
```

*To display all lines containing '**.**'*

```
$grep "\." def
that match a given pattern.
that contain "to". There are
only three such lines.
$        ** '\' is used to ignore special meaning of '.' **
```

*To display all lines ending with '**are**'*

```
$grep "are$" def
that contain "to". There are
$
```

*To display all lines containing '**th**' followed by any lower case alphabet*

```
$grep "th[a-z]" def
Use the grep command to select
and print only those lines
that match a given pattern.
Enter the following command to
print out the lines in 'def'
that contain "to". There are
only three such lines.
$
```

*To display all lines containing '**th**' followed by '**e**' or '**o**' or '**r**'*

```
$grep "th[eor]" def
Use the grep command to select
and print only those lines
Enter the following command to
print out the lines in 'def'
only three such lines.
$
```

Paginate display of files (*pg* command)

This command allows the examination of files one screenful at a time. Each screenful is followed by a prompt **:**. If user press <**Return**> key, another page is displayed; other possibilities are discussed below.

q	quit
h	help
l	next line is displayed
-1	previous page is displayed
+1	next page is displayed

number	**number**ed page is displayed
$	last page is displayed
/pattern/	search forward for pattern
*?***pattern***?*	search backward for pattern
p	display previous file
n	display next file
s *flname*	save current file in flname
! <*cmd***>**	run UNIX command <*cmd*>

*To display file named **file5** page wise*

```
$pg file5
a
b
.
.
x
:
.
.
(EOF):
$
```

To display more than one file page wise

```
$pg file5 file6
a
b
.
.
x
:
.
.
(EOF):
(Next file: file6):
1
2
.
.
:
.
.
(EOF):
$
```

To display file page wise 10 lines at a time

```
$pg -10 file5
a
b
.
.
j
```

```
:
.
.
(EOF):
$
```

*To display file page wise starting from line **5***

```
$pg +5 file5
e
f
.
.
:
.
.
(EOF):
$
```

*To display file page wise and prompting with a user mention **string** and **page no***

```
$pg -p "Page %d" file5
a
b
.
.
Page 1
.
.
(EOF)Page 2
$
```

View a file one screen full at a time (*more* command)

This command allows examination of continuous text one screen full at a time. Each screenful is followed by a prompt **--MORE--**. If user press **<*space bar*>** key, another page is displayed; other possibilities are discussed below.

q	quit
h or **?**	help
<*return*>	display next line of text
s	skip forward 1 line
f	skip forward 1 screen
/<*regular expression*>	search for occurance of regular expression
n	searches for occurance of last regular expression
'	Go to place where previous seach started
=	display current line number
v	start up vi editor at current line
:f	display current file name and line number
:p	go to previous file
:n	go to next file

*To display file named **file5** one screen full at a time*

```
$more file5
a
b
.
.
x
--More--(88%)
.
.
$
```

To display more than one file one screen full at a time

```
$more file5 file6
:::::::::::
file5
:::::::::::
a
b
.
.
--More--(80%)
.
--More--(Next file: file6)
::::::::::::::
file6
::::::::::::::
1
2
.
 --More--
 .
$
```

To display file as 10 lines at a time

```
$more -10 file5
a
b
.
.
j
--More--
.
.
$
```

To display file one screen full at a time starting from line 5

```
$more +5 file5
e
f
```

```
.
.
:
.
.
$
```

To display file one screen full at a time and prompting with a message **Hit space to continue, Del to abort**

```
$more -d file5
a
b
.
.
--More--[Hit space to continue, Del to abort]
.
.
$
```

Pipes:

A *pipe* is a way to connect the output of one program to the input of another program. A *pipeline* is a connection of two or more programs through *pipes*. The vertical bar character **|** tells the shell to set up a **pipeline**.

To print sorted list of users

```
$who | sort
general tty01 Feb 02 12:35
subrata ttyi14 Feb 02 12:42
ranjan ttyi15 Feb 02 12:41
$
```

To look for a particular user

```
$who | grep "subrata"
subrata ttyi14 Feb 02 12:42
$
```

To display directory page wise

```
$ls | pg
abc
cdb
.
.
:        Press any key to see next page
.
.
(EOF)
$
```

To count in how many terminals subrata logged in

```
$who | grep "subrata" | wc -l
1
$
```

To save standard output in a file, as well as display it on the terminal

To print users list and also store them in a file named user

```
$who | tee user
general         tty01           Feb 02 12:35
subrata         ttyi14  Feb 02 12:42
ranjan  ttyi15  Feb 02 12:41
$

$cat user
general         tty01           Feb 02 12:35
subrata         ttyi14  Feb 02 12:42
ranjan  ttyi15  Feb 02 12:41
$
```

To store user list in a file named user, as well as pipe to another process sort to print sorted list of users

```
$who | tee user | sort
general         tty01           Feb 02 12:35
ranjan  ttyi15  Feb 02 12:41
subrata         ttyi14  Feb 02 12:42
$

$cat user
general         tty01           Feb 02 12:35
subrata         ttyi14  Feb 02 12:42
ranjan  ttyi15  Feb 02 12:41
$
```

To print list of users as well as its count on the terminal

```
$who | tee /dev/tty | wc -l
general         tty01           Feb 02 12:35
subrata         ttyi14  Feb 02 12:42
ranjan  ttyi15  Feb 02 12:41
3
$
```

The tee command stores standard output in a file named /dev/tty which is stands for terminal device

To display uniq / duplicate entry

```
$cat number
```

```
100 200 300
200 300 100
300 100 200
100 200 300
300 100 200
$
```

To display one copy of the redundant records

```
$sort number | uniq
100 200 300
200 300 100
300 100 200
$
```

uniq requires a sorted file

To display only uniq records

```
$sort number | uniq -u
200 300 100
$
```

To display only duplicate records

```
$sort number | uniq -d
100 200 300
300 100 200
$
```

To display frequency of occurrence of all records

```
$sort number | uniq -c
2 100 200 300
1 200 300 100
2 300 100 200
$
```

To extract specific columns from a file (fields are separated by ":")

```
$cat country
USSR:8650:262:Asia
Canada:3852:24:North_America
China:3692:866:Asia
USA:3615:219:North_America
Brazil:3286:116:South_America
Australia:2968:14:Australia
India:1269:637:Asia
Argentina:1072:26:South_America
Sudan:968:19:Africa
Algeria:920:18:Africa
$
```

To display 2nd and 3rd fields

```
$cut -d":" -f2,3 country
8650:262
3852:24
3692:866
3615:219
3286:116
2968:14
1269:637
1072:26
968:19
920:18
$
```

To display from 2nd field to 4th field

```
$cut -d":" -f2-4 country
8650:262:Asia
3852:24:North_America
3692:866:Asia
3615:219:North_America
3286:116:South_America
2968:14:Australia
1269:637:Asia
1072:26:South_America
968:19:Africa
920:18:Africa
$
```

To display 5th to 20th character

```
$cut -c5-20 country
:8650:262:Asia
da:3852:24:North
a:3692:866:Asia
3615:219:North_A
il:3286:116:Sout
ralia:2968:14:Au
a:1269:637:Asia
ntina:1072:26:So
n:968:19:Africa
ria:920:18:Afric
$
```

To paste more than one file horizontally

```
$cat num
100 200 300
200 300 100
```

```
300 100 200
$

$cat a1
          BEGIN {OFS=":"}
                    {print $1,$2,$3,$4}
$

$paste num a1
100     200     300     BEGIN {OFS=":"}
200     300     100               {print $1,$2,$3,$4}
300     100     200
$
```

Translating Characters

To change lower case letters with upper case letters

```
$who | tr '[a-z]' '[A-Z]'
GENERAL      TTY01        FEB 02 12:35
SUBRATA      TTYI14       FEB 02 12:42
RANJAN       TTYI15       FEB 02 12:41
$
```

To squeeze multiple consecutive occurrences of its argument to a single character (to compress multiple blank spaces to a single space

```
$who | tr -s ' ' | tee user
general           tty01             Feb 02 12:35
subrata           ttyi14   Feb 02 12:42
ranjan   ttyi15   Feb 02 12:41
$
```

To replace each blank character with a ':' character

```
$cat user | tr ' ' ':' | tee user1
general:tty01:Feb:02:12:35
subrata:ttyi14:Feb:02:12:42
ranjan:ttyi15:Feb:02:12:41
$
```

To delete the character : from the file named user1

```
$tr -d ':' < user2
generaltty01Feb021235
subratattyi14Feb021242
ranjanttyi15Feb021241
$
```

The command tr does not accept a filename as an argument, instead input has to be redirected (<) or can be supplied through a pipe

Group Commands:

A command usually ends with the newline character at the end of the command. However, several commands can also be grouped on one line by using a semi-colon (:) as a command separator. The semicolon defines end of command. For example :

```
$date;who
Wed Feb 02 11:43:34 IST 1994
general         tty01           Feb 02 12:35
subrata         ttyi14   Feb 02 12:42
ranjan  ttyi15   Feb 02 12:41
$
```

To count no. of lines in above result

```
$(date;who) | wc -l
4
$
```

Background Process:

It is common experience some commands take a long tie to complete execution. To overcome this problem, UNIX provides a method of running commands in the background while the user continues working in the foreground. For instance, user want to sort a big file on background and execute ls command in foreground immediately.

```
$sort abc > sdf &
969
$ls
.
.
.
```

** **&** indicates command will be executed in background **
** **969** indicates process number **

To display status of all processes

```
$ps
PID TTY TIME COMMAND
894 01 0:01 sh
969 01 0:01 sort
972 01 0:00 ps
$
```

To finished all background processes before execute another command

```
$wait
$
```

To kill a background process

```
$kill 969
$ps
PID TTY TIME COMMAND
894 01 0:01 sh
972 01 0:00 ps
$
```

To give more than one name to a file (link)

```
$ls -l
total 7
-rw-r--r-- 1 general group 23 Feb 02 12:48 abc
-rw-r--r-- 1 general group 23 Feb 02 14:48 arc
-rw-r--r-- 1 general group 228 Feb 02 02:49 def
-rw-r--r-- 1 general group 52 Feb 02 04:48 file5
-rw-r--r-- 1 general group 18 Feb 02 12:49 xyz
$
```

```
$ln abc cdb
$ls -l
total 9
-rw-r--r-- 2 general group 23 Feb 02 12:48 abc
-rw-r--r-- 2 general group 23 Feb 02 12:48 cdb
-rw-r--r-- 1 general group 23 Feb 02 14:48 arc
-rw-r--r-- 1 general group 228 Feb 02 02:49 def
-rw-r--r-- 1 general group 52 Feb 02 04:48 file5
-rw-r--r-- 1 general group 18 Feb 02 12:49 xyz
$
```

To remove the file/link

```
$rm cdb
$ls -l
total 7
-rw-r--r-- 1 general group 23 Feb 02 12:48 abc
-rw-r--r-- 1 general group 23 Feb 02 14:48 arc
-rw-r--r-- 1 general group 228 Feb 02 02:49 def
-rw-r--r-- 1 general group 52 Feb 02 04:48 file5
-rw-r--r-- 1 general group 18 Feb 02 12:49 xyz
$
```

To send files in line printer

To send in defaul line printer

```
$lp abc
request id is pr0-001 (1 file)
$
```

To send in specified line printer

```
$lp -d pr1 abc
request id is pr0-002 (1 file)
$
```

To suppress the banner

```
$lp -o nobanner abc
request id is pr0-003 (1 file)
$
```

To print more than one copy of the request

```
$lp -n2 ab*
request id is pr0-004 (3 file)
$
```

To display status of line printers

To display list of printers and default printer

```
$lpstat -s
system default destination: pr0
device for pr0: /dev/lp0
device for pr1: /dev/lp1
$
```

To display all status information

```
$lpstat -t
scheduler is running
system default destination: pr0
device for pr0: /dev/lp0
device for pr1: /dev/lp1
pr0 accepting requests since Thu Dec 23 14:31:39 1993
pr1 accepting requests since Fri Dec 24 10:30:00 1993
printer pr0 now printing pr0-002. enabled since Mon 14
12:09:34 1994. available.
printer pr1 is idle. enable since Mon 14 14:30:16 1994.
available.
pr0-002 general 75 Mar 15 12:24 on pr0
pr0-003 general 124 Mar 15 12:26
$
```

To cancel printer request

To cancel current request

```
$cancel pr0
request "pr0-002" cancelled
$
```

To cancel specified request

```
$cancel pr0-003
request "pr0-003" cancelled
$
```

To display all terminal characteristics

```
$stty -a
speed 9600 baud; ispeed 9600 baud; ospeed 9600 baud;
line = 0; intr = DEL; quit = ^\; erase = ^H; kill = ^U; eof = ^D;
eol = ^@; swtch = ^@;susp <undef>;start = ^Q;stop = ^S;
.
.
$
```

To set <ctrl - c> as interrupt key instead of <*Del*>

```
$stty intr \^c
$
$stty -a
speed 9600 baud; ispeed 9600 baud; ospeed 9600 baud;
line = 0; intr = ^C; quit = ^\; erase = ^H; kill = ^U; eof = ^D;
eol = ^@; swtch = ^@;susp <undef>;start = ^Q;stop = ^S;
.
.
$
```

To set <*ctrl - a*> as eof character instead of <ctrl - d>

```
$stty eof \^a
$
$stty -a
speed 9600 baud; ispeed 9600 baud; ospeed 9600 baud;
line = 0; intr = ^C; quit = ^\; erase = ^H; kill = ^U; eof= ^A;
eol = ^@; swtch = ^@;susp <undef>;start = ^Q;stop = ^S;
.
.
$
```

To display ASCII octal value of any file contents (to see non-printable character)

```
$cat a1
sdlkf
^D^D^D^D^D
^D
$
$od -b a1
0000000 163 144 154 153 146 012 004 004 004 004 004 012 004 012
0000016
$
```

each displays sizteen bytes of data in ASCII octal format.

To display amount off free space available on the disk

```
$df
/ (/dev/root ): 27762 blocks 15231 i-nodes
$
```

To display total and free space available on the disk

```
$df -t
/ (/dev/root ): 27762 blocks 15231 i-nodes
total: 174338 blocks 21792 i-nodes
$
```

Communication

The Bulletin Board (news command)

This command is invoked by user to read message that is stored in a file in the directory /usr/news. This messages can be stored by any user.

To store message

```
$cat > /usr/news/mesg1
Hello everybody !!
^d
$
```

To read message

```
$news
mesg1 (subrata) Wed Feb 10:43:34 IST 1994
Hello everybody !!
$
```

To display only the filenames of those messages which have not been displayed

```
$news -n
news: mesg1 mesg2
$
```

To display specific message

```
$news mesg2
mesg2 (general) Wed Feb 10:49:34 IST 1994
Good Morning everybody !!
$
```

To lists out the number of news items which have still not been read

```
$news -s
2 news items
$
```

To displays the contents of all news items regardless of whether they have been read or not

```
$news -a

mesg1 (subrata) Wed Feb 10:43:34 IST 1994

Hello everybody !!

mesg2 (general) Wed Feb 10:49:34 IST 1994

Good Morning everybody !!
```

```
$
```

Two way communication (*write command*)

This command lets user has a two-way communication with any user who is currently logged in. One user writes his message and then waits for the reply from the other. In this way it is possible to continue a conversation until such time as one or both the users press ^d to terminate it.

To starts a dialogue with ranjan from subrata

```
$write ranjan
Good Morning
```

To reply subrata's message

```
write subrata
Morning
```

When subrata is logged in to more than one terminal, say on tty04 and tty07, then the command

```
write subrata
Morning
```

sends message to the terminal with the lowest number, i.e. tty04. But if user really want to send message to the terminal tty07, then the command is

```
write subrata tty07
```

Suppose subrata is logged in to more than one terminal, say on tty04 and tty07. To comminucate between these two terminals always mention the terminal_number with login_name.

In terminal tty04

```
$write subrata tty07
Good Morning
```

In terminal tty07

```
write subrata tty04
Morning
```

To send message from a file

```
$write ranjan < messg.dat
$
```

However, if ranjan is not logged in, then the system responds with an error message:

```
ranjan is not logged in
```

Insulation from other users (mesg command)

Communication, single or two-way, can be disconnecting to a user who is presently busy with his own job.

To prevent other users from writing to his terminal

```
$mesg n
```

To enable receipt of such writing

```
$mesg y
```

Using the Mail Box (mail command)

This command is the UNIX repertoire of electronic mail. Unlike write command, it enables sending of mail to a user even if he is not logged in. It is mainly used for non-interactive communication. It saves the message in a mailbox, which normally is placed in the directory /usr/mail, and has the same name as the login name. For instance, /usr/mail/ranjan. In SCO UNIX, the directory is /usr/spool/mail.

To send a mail to the user subrata from ranjan

```
$mail subrata
Subject: Morning

Good Morning
How are you ?
^d
$
```

The message does not directly appear on the receiver's terminal either. When the receiver next time log in the following message will be flashing in his/her terminal.

```
You have mail
```

To receive a mail

```
$mail
>N 3   ranjan   Wed Feb 02 11:55 10/260 Morning
N 2    general       Tue Feb 01 14:35 15/459 Meeting
U 1    root          Tue Feb 01 10:12 23/789 Notice
&
```

☞ A greater than sign (>) pointing to the current message
☞ A status indicator: **N** for new and **U** for unread
☞ The number of the message
☞ The sendor
☞ The date sent
☞ The number of lines and characters
☞ The subject (if the message contains a Subject:)

To read 3rd message

```
&3
Message 3:
From ranjan Wed Feb 02 11:55:45 IST 1994
To: subrata
Subject: Morning

Good Morning
How are you ?
&
```

Other commands at the & prompt

-	prints the previous message
h	prints header of all messages
d n	deletes message n (the current message if n not specified)
u n	undeletes message n (the current message if n not specified)
r *user*	replies to specific sender (the sender only when user is not specified); deletes message after reply.
R	sends a reply to everyone on the distribution list
m *user*	mail to specific user
l n	sends the current message n to the line printer (the current message if n not specified)
s *flname*	saves current message with headers in file flname (the file mbox in the login directory in case flname is not specified)
w *flname*	saves current message without headers in file flname (the file mbox in the login directory in case flname is not specified)
q	quits from mail
! *<cmd>*	runs Unix command *<cmd>*
sh	enters a new shell
?	give helps

Sending a message to self (calendar command)

This command searches a text file calender in the current directory for lines containing either today or tomorrow's date. It then displays the matched lines. This is a sort of an engagement diary which consists of textual information.

```
$cat calendar
Principal visiting the department on Feb 03, 1994
Prof. S. Kar. take class on shell programming on 02/04/94
A practical class of shell programming on 02/05/94
Validiction on March 02
$

$date
Fri Feb 4 12:10:15 IST 1994
$

$calendar
```

```
Prof. S. Kar. take class on shell programming on 02/04/94
A practical class of shell programming on 02/05/94
$
```

<u>Disk & Tapes</u>

<u>To format disks</u>

To take backup, user must format the floppy disk using format command. In general, catridge tapes do not need to be formatted. However, tapes that are to be used with mini drives must be formatted.

To format a 5.25" 360KB disk

```
$format /dev/rfd048ds9
```

To format a 5.25" 720KB disk

```
$format /dev/rfd096ds9
```

To format a 5.25" 1.2MB disk

```
$format /dev/rfd095ds15
```

To format a 3.5" 720KB disk

```
$format /dev/rfd0135ds9
```

To format a 3.5" 1.44MB disk

```
$format /dev/rfd0135ds18
```

- **r** indicates a raw (character) interface to the disk
- **fd0** indicates floppy drive number (0,1,2,etc.)
- **48**
  /96
  /135 indicates no. of tracks per inch
- **ds** indicates double sided (ss - single sided)
- **9**
  /15
  /18 indicates no. of sectors per track

To format mini cartridge tapes

```
$mcart format device_name
```

<u>To copy input output (cpio command)</u>

The **cpio** (<u>co</u>py <u>i</u>nput <u>o</u>utput) can be used to copy files to and from a backup device. This backup device can be a magnetic or cartidge tape, a floppy disk, or even a disk file.

cpio uses the standard input to take the list of filenames. It then copies these files to the standard output. Since it uses standard input and standard output, it is always used with the shell's redirection and piping sysmbols.

To dumps the contents of the current directory to the disk file

```
$ls | cpio -ov > archfile
```

-o (copy out) reads the standard input to obtain a list of filenames and copies those files onto the standard output.**v** (*verbose*) option display the name of each file that is being created

To dumps the contents of the current directory to the floppy disk

```
$ls | cpio -ov > /dev/fd048ds9
```

To dumps the contents of the current directory except ***.c** *files to the floppy disk*

```
$ls | cpio -ovf "*.c" > /dev/fd048ds9
```

To dumps the contents of the current directory as well as its sub-directories to the floppy disk

```
$find . -print | cpio -ov /dev/fd048ds9
```

To restore all files from floppy disk

```
$cpio -iv < /dev/fd048ds9
```

-i (**copy input**) extracts files from the standard input, which is assumed to be the product of a previous cpio -o

To restore specific files from floppy disk

```
$cpio -iv "*.c" < /dev/fd048ds9
```

To restore all files as well as sub-directories and all files from these sub-directories

```
$cpio -ivd < /dev/fd048ds9
```

To display the contents of the archive file [from device] without restoring the files

```
$cpio -it < dev/fd048ds9
```

To take backup on disk or tape (*tar* command)

The tar (tape archive) command saves and restores files to and from an archive medium, which is typically a floppy disk or tape.

To take backup of all files from dir1 directory to disk

```
$tar -cvf /dev/fd048ds9 dir1
```

-c instructs tar command to create a new backup and
overwrite the old one if exists.
v (verbose) causes it to display the each file names it treats.
f indicates next argument as the name of the archive [device]

To display files from backup cartidge tape

```
$tar -tvf /dev/rct0
```

To extract files from backup disk to current directory

```
$tar -xvf /dev/fd048ds9
```

To add more files on a backup disk

```
$tar -uvf /dev/fd048ds9 test1.c
```

u option cannot be used with the tape archive.

As a System Administrator

To address all users (*wall* command)

```
$wall
The machine will be shut down today at 12:20 hrs.
^d
$
```

To create a file system (*mkfs* command)

A filesystem is a distinct division of the operating system, consisting of files, directories, and the information needed to locate and access them.

To create a file system on a floppy disk, it needs to be first formatted with format command and after that, the mkfs (make file system) command can be used like follows

```
#mkfs /dev/rfd048ds9 720
```

360KB = 360000B = 360000/512Block = 720Block

To engage and disengage a file system (*mount* and *umount* command)

Once a file system has been created, it is needed to add it to the main file system. To do that, user must indicate the branch of the main file system which is to be attached to the root directory of new file system. This process is called mounting, and achieved with the mount command.

To mount 360KB floppy disk in first floppy drive

```
#mkdir ownfs
#cd ownfs
#pwd
/ownfs
#mount /dev/fd048ds9 /ownfs
#mount
/ on /dev/root read/write on Fri Feb 04 14:34:55 1994
/ownfs on /dev/fd048ds9 read/write on Mon Feb 07 12:10:05 1994
```

After this device is mounted, its root directory now becomes the /ownfs.

To dismount the floppy disk from the main file system

```
#umount /ownfs
```

*To change file ownership (**chown** and **chgrp** commands)*

When a file is created, the user becomes the owner of the file, and the group to which the user belongs becomes the group owner. If user is not the owner of a file, he simply can not change the major file attributes. For instance, you can neither edit the file, nor change its permissions.

To change file owner of the file item

```
#ls -l item
-rwxrw---x 1 general group 10 Feb 01 10:30 item
#chown ranjan item
#ls -l item
-rwxrw---x 1 ranjan group 10 Feb 01 10:30 item
```

To change group owner of the file item

```
#ls -l
-rwxrw---x 1 ranjan group 10 Feb 01 10:30 item
#chgrp bin item
#ls -l item
-rwxrw---x 1 ranjan bin 10 Feb 01 10:30 item
#
```

Account Management

All user accounts are stored in the file called **/etc/passwd**. This file is very important and if destroyed no one will be able to login to the system. Each line of the /etc/passwd file corresponds to an account. The fields of the /etc/passwd file are as follows (delimited by :)

- **Account name** This is the unique user login name
- **Password** If the account has a password, its encrypted form would be stored. If not present this field will be blank
- **User id** The user id number is 0 for superuser, less than 100 for all system accounts, and in between 100 and 30000 for user accounts
- **Group id** Each account belongs to a group. This group membership defines collective permissions in files
- **Comment field** This normally contains more information about the account
- **Home directory** The full pathname of the home directory of the account is in this field
- **Start up program** This is the program invoked immediately when the account accesses the system. For ordinary accounts it is normally a shell program

A sample of ***/etc/passwd*** *file*

```
root:muZ:0:1:Superuser:/:
general:B09..,er:200:50:General Users:/usr/general:/bin/sh
ranjan:hoopK9:201:50:Ranjan Dasgupta:/usr/ranjan:/bin/sh
subrata:Y09oiu:202:50:Subrata Baguli:/usr/subrata:/bin/csh
```

Each user account in the UNIX system belongs to a group. This grouping facilitates the collective ownership of files or processes. If the group permissions on a file are set then any ser of the group can access the file. The **/etc/passwd** file defines the default group for each of its accounts. The /etc/group file describes the groups in the system. Each line of the /etc/group file corresponds to one group with the following fields delimited by a colon (:)

☞ **Group name**	Name of the group
☞ **Password**	Password of the group in encrypted form. Normally it is left as blank.
☞ **Group id**	A unique integer less than 100 for system groups, 100 - 30000 for user groups
☞ **User id's**	Account names that are belong to this group separated by commas (,)

A sample of ***/etc/group*** *file*

```
root::0:
other::1:root
bin::2:bin
group::50:general,ranjan,subrata
```

To add an account

The steps are:

☞ Add an entry in the /etc/passwd file giving appropriate values to each field
☞ Leave the second field (password) blank
☞ Save the /etc/passwd file
☞ Create the appropriate home directory for the new account.
☞ Change the owner of the directory to the account name
☞ Change the group of the directory to the account's group
☞ Add account name to the appropriate group entry in the /etc/group file
☞ Save the /etc/group file
☞ Copy the /etc/profile file into the home directory as .profile
☞ Change the owner and group of the .profile file.

To remove an account

The steps are:

☞ Backup all the files from the appropriate home directory
☞ Remove all files and directories of the account
☞ Delete the appropriate entry from /etc/passwd file
☞ Delete the account name from appropriate entry of /etc/group file

To set working environments

Whenever any user logs in the shell program starts up and executes commands in the file **/etc/profile** and those in a file called .profile in the accounts home directory. Every user can edit the .profile file in his/her home directory to include any system requirement.

The following some of the shell environment variables that get initialised the moment the user logs into the system

HOME	Home directory
MAIL	Mailfile pathname
LOGNAME	Account name
PATH	Command seach path
PS1	Primary prompt string
PS2	Secondary prompt string

*A sample of **.profile** file*

```
PATH=$PATH:$HOME
MAIL=/usr/spool/mail/`logname`
PS1='General> '
export PATH MAIL PS1
```

Thus using the variables and export (makes the environment of the parent process available in the child process) command in the .profile file any user can customise his/her working environment. The present values of various environment variables can seen by either env or set command.

Visual Editor

To invoke *VI* editor

$vi file name

To insert and append

i Insert at cursor position
a Append to right of the cursor
A Append at the end of line

Cursor movement commands

h Moves cursor one character to the left
l Moves cursor one character to the right
j Moves cursor down by one line

k Moves cursor up by one line
w Moves cursor to the next word
b Moves cursor backwards to previous word

$ Moves cursor to the end of a line
0 Moves cursor to the begining of a line

{ Moves cursor to the begining of a paragraph
} Moves cursor to the end of a paragraph

^**b** Scroll the screen up by 24 lines
^**f** Scroll the screen down by 24 lines

:n Go to line number n
:+n To moves cursor n lines forward
:-n To moves cursor n lines backward

Delete commands

x Delete character
dw Delete word
dd Delete line

Block operation commands

*n***yy** Yank n lines into buffer
p Put back yanked text after cursor
P Put back yanked text before cursor

Searching commands

/*pattern*/ Find the next line containing the pattern
?*pattern***?** Find the previous line containing the pattern
n Find the next occurance of the pattern

Substitute text

s*text* Substitute text for character till <ESC> key is pressed
cw*text* Substitute text for word till <ESC> key is pressed
:m,ns/*x*/*y*/ Substitute y for occurance of x from line m to line n
:1,$s/*x*/*y*/ Substitute y for occurance of x across the file

Saving and Exiting from *VI*

:w Save all changes made so far
:wq Save all changes and quit from VI
:q Quit (if no changes made)
:q! Quit without saving

Other Functions

:nu To display current line number
:se nu To turn on automatic line numbering
:se nonu To turn off automatic line numbering
:!<*cmd***>** Execute a <cmd> unix shell command
:r *flname* To read file flname into present cursor location

The Bourne Shell Programming

A simple shell procedure that prints information regarding systems environment

```
$cat b1
echo "The current date and time: \c" date
echo "The number of users: \c" who|wc -l
echo "Personal Status: \c" who am i
$

$sh b1
The current date and time: Thu Feb 10 13:27:05 IST 1994
The number of users: 1
Personal Status: general ttyi15 Feb 10 10:53
$
```

Handling variables in shell procedure

```
$cat b2
TIME="The current date and time: \c"
USERS="The number of users: \c"
ME="Personal Status: \c"
echo "$TIME"
date
echo "$USERS"
who|wc -l
echo "$ME"
who am i
$

$sh b2
The current date and time: Thu Feb 10 13:27:47 IST 1994
The number of users: 1
Personal Status: general ttyi15 Feb 10 10:53
$
```

Enter a user name and check it is a valid user or not

```
$cat b3
echo "Enter a user name: \c"
read NAME
if grep "^$NAME:" /etc/passwd > /dev/null
then
echo "'$NAME' is a valid user"
else
echo "'$NAME' is not a valid user"
fi
```

Here, output of grep command will go to the '**/dev/null**' file

```
$
 $sh b3
Enter a user name: sanjay
'sanjay' is not a valid user
$
```

Handling command line arguments

```
 $cat b4
```

```
if test $# -ne 1
'test' command is used to test then
for status of files and values
 echo "Usage: $0 <user name>" of variables. The output of
 exit 'test' command is always true
 fi or false
```

```
if grep "^$1:" /etc/passwd > /dev/null
then
echo "'$1' is a valid user"
else
echo "'$1' is not a valid user"
fi
$
```

$# indicates total no. of arguments
$0,$1,...$9 indicates 1st,2nd,..10th argument
-ne stands for not_equal_to

```
$sh b4 subrata
'subrata' is a valid user
$sh b4 sanjay
'sanjay' is not a valid user

$
```

Handling logical operators

Enter a number as first argument and determine a grade for it according to following rule

```
Number Grade
------ -----
90 - 100 E+
80 - 089 E
70 - 079 A+
60 - 069 A
50 - 059 P
49 - 000 F
```

```
$cat b5
if test $# -ne 1
then
echo "Usage: $0 <numbre>"
exit
fi

if test $1 -gt 100 -o $1 -lt 0
then
echo "\nInvalid number"
exit
fi


if test $1 -ge 90
then
gd="E+"
elif test $1 -ge 80
then
gd="E"
elif test $1 -ge 70
then
gd="A+"
elif test $1 -ge 60
then
gd="A"
elif test $1 -ge 50
then
gd="P"
else
gd="F"
fi

echo "\nGrade is $gd ($1 %)"
```

\n stands for carriage return

```
$

$sh b5 98
Grade is E+ (98 %)
$
```

Determine file type, Enter file name as first argument

```
$cat b6
if test $# != 1
then
echo "usage: $0 <file name>"
exit
fi
```

```
if test -d $1
then
echo "directory file"
elif test -f $1
then
echo "ordiary file"
else
echo "$1 not found"
exit
fi

if test -r $1
then
echo "readable"
fi

if test -w $1
then
echo "writeable"
fi
if test -x $1
then
echo "executable"
fi
$

$sh b6 b6
ordiary file
readable
writeable
$ls -l b6
-rw-r--r-- 1 general group 348 Feb 10 11:24 b6
$
```

Validate no. of users entered as arguments

1st method

```
$cat b7
for name in $* ## $* returns all command line arguments do
if grep "^$name:" /etc/passwd > /dev/null then
echo "'$name' is a valid user"
else
echo "'$name' is not a valid user"
fi
done
$

$sh b7 a subrata b ranjan
'a' is not a valid user
'subrata' is a valid user
'b' is not a valid user
```

```
'ranjan' is a valid user
$
```

Validate no. of users entered as arguments

2nd method

```
$cat b8
until test -z "$1" ## Continue until $1 is not blank
do
if grep "^$1:" /etc/passwd > /dev/null
then
echo "'$1' is a valid user"
else
echo "'$1' is not a valid user"
fi
shift ## shift $2 as $1, $3 as $2 and so on
done
$

$sh b8 a subrata b ranjan
'a' is not a valid user
'subrata' is a valid user
'b' is not a valid user
'ranjan' is a valid user
$
```

Display contents of all files in the current directory after confirmation

```
$cat b9
for fl in `ls`
do
echo "$fl? \c"
read rl
if test "$rl" = y -o "$rl" = Y
then
cat $fl
fi
done
$

$sh b9
b1? y
echo "The current date and time: \c"
date
echo "The number of users: \c"
who|wc -l
echo "Personal Status: \c"
who am i
b10? n
$
```

Validate a user after entering its name and the process is still continue until ^d or del key is pressed

```
$cat b10
echo "User Name: \c"

while read name
do
if grep "^$name:" /etc/passwd > /dev/null
then
echo "'$name' is a valid user"
else
echo "'$name' is not a valid user"
fi
echo "\nUser Name: \c"
done
$

$sh b10
User Name: subrata
'subrata' is a valid user

User Name: sanjay
'sanjay' is not a valid user

User Name: ^d
$
```

A user friendly version of chmod. The first argument should be *r*, *w* or *x* and second argument is a file name

```
$cat b11
if test $# -ne 2
then
echo "Usage: $0 <permission> <file name>"
exit
fi

if test -f $2 -o -d $2
then
case $1 in
r)chmod u+r $2;;
w)chmod u+w $2;;
x)chmod u+x $2;;
*)echo "No such permission"
esac
else
echo "'$2' not found"
fi
$

$ls -l b11
```

```
-rw-r--r-- 1 general group 331 Feb 10 12:16 b11
$sh b11 x b11
$ls -l b11
-rwxr--r-- 1 general group 331 Feb 10 12:16 b11
$
```

Arithmetic operation in shell

```
$cat b12
echo "Enter value of 'a': "
read a
echo "Enter value of 'b': "
read b
c=`expr $a + $b`
echo "'a + b' is equal to $c"
c=`expr $a - $b`
echo "'a - b' is equal to $c"
c=`expr $a \* $b`
echo "'a * b' is equal to $c"
c=`expr $a / $b`
echo "'a / b' is equal to $c"
$
```

Here, ***** is preceded by \ so that it is not interpreted as wildcard character by the shell script during execution of the **expr** command

```
$sh b12
Enter value of 'a': 7
Enter value of 'b': 3
'a + b' is equal to 10
'a - b' is equal to 4
'a * b' is equal to 21
'a / b' is equal to 2
$
```

Menu driven procedure *first process*

```
$cat b13
while true
do
tput clear ## clear the screen
tput cup 10 30 ## position the cursor at 10th row
echo "System Status" ## 30th column
tput cup 12 30
echo "1. Todays date"
tput cup 13 30
echo "2. Total no. of users"
tput cup 14 30
echo "3. Personal Status"
tput cup 15 30
echo "4. Quit"
```

```
tput cup 17 30
echo "Enter choice(1-4): \c"
read ch

tput clear
tput cup 12

case $ch in
1) echo "The current date and time: `date`" ;;
2) echo "Total number of users: `who|wc -l`" ;;
3) echo "Personal Status: `who am i`" ;;
4) break ;;
*) echo "Invalid Choice"
esac

sleep 5
done
$

$sh b13
System Status

1. Todays date
2. Total no. of users
3. Personal Status
4. Quit

Enter choice(1-4): 1

The current date and time: Thu Feb 10 13:39:44 IST 1994
.
.
System Status

1. Todays date
2. Total no. of users
3. Personal Status
4. Quit

Enter choice(1-4): 4
$
```

Menu driven procedure *second process*

```
$cat b14
while true
do
tput clear ## clear the screen
tput cup 8 ## position the cursor at 8th row
## 1st column
cat << TS
System Status
```

```
 1. Todays date
 2. Total no. of users
 3. Personal Status
 4. Quit

 Enter choice(1-4):
 TS
 tput cup 15 47
 read ch

 tput clear
 tput cup 12

 case $ch in
 1) echo "The current date and time: `date`" ;;
 2) echo "Total number of users: `who|wc -l`" ;;
 3) echo "Personal Status: `who am i`" ;;
 4) break ;;
 *) echo "Invalid Choice"
 esac

 sleep 2 # wait for 2 seconds
 done
```

cat << TS **User can display a text. with the help of '*here document*'.**
system as used in this example TS
$

```
 $sh b14
.
 System Status

 1. Todays date
 2. Total no. of users
 3. Personal Status
 4. Quit

 Enter choice(1-4): 1

.
The current date and time: Thu Feb 10 13:39:44 IST 1994
.
.
 System Status

 1. Todays date
 2. Total no. of users
 3. Personal Status
 4. Quit

 Enter choice(1-4): 4
```

```
$
```

Display contents of all ordinary files in the current directory after confirmation

```
$cat b15
for fl in `ls`
do if test -f $fl
then
echo "$fl? \c"
read rl
if test "$rl" = y -o "$rl" = Y
then
cat $fl
fi
fi
done
$

$sh b15
b1? y
echo "The current date and time: \c"
date
echo "The number of users: \c"
who|wc -l
echo "Personal Status: \c"
who am i
b10? n
$
```

Simple Programming with Awk

Awk (Aho Weinberger Kernighan) filter reads from a file one record at a time. Records are ended with a newline character and fields of any record are recognised by blank or tab character (by default). This reading of file is sequential in nature and depending upon the program associated with the awk filter, iit outputs the records (a part of records) which satisfy the conditions mentioned in the program.

The general structure of awk is

awk '*program*' filenames ...

The program is

pattern {*action*}
...

Where **filename** represents the text file on which awk work and program represents the condition which is to be checked with the records of the file and accordingly the output awk is given. This program can be constituted in two ways - one by its own syntax (explained in subsequent section) and also it accepts C language like format (i,e. printf, for loop, etc.)

The *fields* of the records are represented by **$1, $2,...**etc. **$0** represents the record as a whole.

Awk has its own set of operators, awk variables and switches explained later.

To explore awk programming create the country file, which contains information about the 10 largest countries in the world. Each record contains the name of country, its area in thousands of square miles, its population in millions, and the continent on which it is found.

```
$cat country
USSR 8650 262 Asia
Canada 3852 24 North_America
China 3692 866 Asia
USA 3615 219 North_America
Brazil 3286 116 South_America
Australia 2968 14 Australia
India 1269 637 Asia
Argentina 1072 26 South_America
Sudan 968 19 Africa
Algeria 920 18 Africa
$
```

To print 1st & 2nd field of the country file

```
$awk '{print $1,$2}' country
USSR 8650
Canada 3852
China 3692
USA 3615
Brazil 3286
Australia 2968
India 1269
Argentina 1072
Sudan 968
Algeria 920
$
```

$1 represents 1st field & $2 represent 2nd field

To print 1st & 3rd field of the country file according to specific format

```
$awk '{printf "%10s %6d\n",$1,$3}' country
USSR 262
Canada 24
China 866
USA 219
Brazil 116
Australia 14
India 637
Argentina 26
Sudan 19
Algeria 18
```

```
$
```

%10s indicate minimum of 10 characters string
%6d indicate minimum of 6 digits integer number

```
$awk '{printf "%-10s %6d\n",$1,$3}' country
USSR 262
Canada 24
China 866
USA 219
Brazil 116
Australia 14
India 637
Argentina 26
Sudan 19
Algeria 18
$
```

%-10s indicate left justified minimum of 10 characters string

To print records in which the 4th field equals the string "*Asia*"

```
$awk '$4=="Asia"' country
USSR 8650 262 Asia
China 3692 866 Asia
India 1269 637 Asia
$
```

To print 1st & 3rd field of each records in which 4th field equals the string "*Asia*"

```
$awk '$4=="Asia" {print $1,$3}' country
USSR 262
China 866
India 637
$
```

To print records in which 4th field not equals the string "*Asia*"

```
$awk '$4!="Asia"' country
Canada 3852 24 North_America
USA 3615 219 North_America
Brazil 3286 116 South_America
Australia 2968 14 Australia
Argentina 1072 26 South_America
Sudan 968 19 Africa
Algeria 920 18 Africa
$
```

To print records in which 2nd field greater than equal to *3000*

```
$awk '$2>=3000' country
```

```
USSR 8650 262 Asia
Canada 3852 24 North_America
China 3692 866 Asia
USA 3615 219 North_America
Brazil 3286 116 South_America
$
```

To print records in which 1st field greater than the string "*S*"

```
$awk '$1 > "S"' country
USSR 8650 262 Asia
USA 3615 219 North_America
Sudan 968 19 Africa
$
```

To print records starting with the first record that contains the string "*USA*", up through the next occurance of the string "*Sudan*"

```
$awk '/USA/,/Sudan/' country
USA 3615 219 North_America
Brazil 3286 116 South_America
Australia 2968 14 Australia
India 1269 637 Asia
Argentina 1072 26 South_America
Sudan 968 19 Africa
$
```

To print records those are starting with string "*A*"

```
$awk '/^A/' country
Australia 2968 14 Australia
Argentina 1072 26 South_America
Algeria 920 18 Africa
$
```

To print records those are starting with string "*A*" and followed by any character between "*l*" and "*r*"

```
$awk '/^A[l-r]/' country
Argentina 1072 26 South_America
Algeria 920 18 Africa
$
```

The grep command that will perform the same

```
$grep "^A[l-r]" country
Argentina 1072 26 South_America
Algeria 920 18 Africa
$
```

To print records in which fourth field contains the string "*America*"

```
$awk '$4~/America/' country
Canada 3852 24 North_America
USA 3615 219 North_America
Brazil 3286 116 South_America
Argentina 1072 26 South_America
$
```

To print records in which 4th field does not contain the string "*America*"

```
$awk '$4!~/America/' country
USSR 8650 262 Asia
China 3692 866 Asia
Australia 2968 14 Australia
India 1269 637 Asia
Sudan 968 19 Africa
Algeria 920 18 Africa
$
```

To print records in which 1st field end with the string "*n*"

```
$awk '$1~/n$/' country
Sudan 968 19 Africa
$
```

To print records in which 4th field contains the string "*Asia*" or "*America*"

```
$awk '$4~/(Asia|America)/' country
USSR 8650 262 Asia
Canada 3852 24 North_America
China 3692 866 Asia
USA 3615 219 North_America
Brazil 3286 116 South_America
India 1269 637 Asia
Argentina 1072 26 South_America
$
```

To print records in which 4th field equal to the string "*Asia*" or "*America*"

```
$awk '$4=="Asia" || $4=="America"' country
USSR 8650 262 Asia
China 3692 866 Asia
India 1269 637 Asia
$
```

To print records in which 4th field equal to the string "*Asia*" and 3rd field greater than *200*

```
$awk '$4=="Asia" && $3>200' country
USSR 8650 262 Asia
China 3692 866 Asia
```

```
India 1269 637 Asia
$
```

Awk also support to present the **program** to awk from a file
awk -f program_file filename ...

To print countries of Asia with a heading

```
$cat a1
BEGIN { print "Countries of Asia"}
$4 ~/Asia/ { print " ", $1}
$
```

BEGIN is a special pattern, execute before first record is read

```
$awk -f a1 country
Countries of Asia
USSR
China
India
$
```

To print countries outside the "*Asia*"

```
$cat a2
BEGIN { print "Countries of Asia"}
$4 !~/Asia/ { print " ", $1}
$

$awk -f a2 country
Countries of Asia
Canada
USA
Brazil
Australia
Argentina
Sudan
Algeria
$
```

To print record number followed by record

```
$cat a3
{ printf "%4d %s\n", NR, $0 }
$
```

NR gives last record number

```
$awk -f a3 country
1 USSR 8650 262 Asia
2 Canada 3852 24 North_America
3 China 3692 866 Asia
4 USA 3615 219 North_America
```

```
5 Brazil 3286 116 South_America
6 Australia 2968 14 Australia
7 India 1269 637 Asia
8 Argentina 1072 26 South_America
9 Sudan 968 19 Africa
10 Algeria 920 18 Africa
$
```

To calculate and print total and average of population

```
$cat a4
BEGIN {s=0}
{ s=s+$3 }
END {print "Total population is", s, "million"
print "Average population of", NR, "countries is", s/NR}
$
```

BEGIN and **END** pattern execute before the first record is read and after the last record is processed.
'**s**' is a user defined variable

```
$awk -f a4 country
Total population is 2201 million
Average population of 10 countries is 220.1
$
```

To print records with heading and also calculate and print total area and population

```
$cat a5
BEGIN { area=0; popu=0
printf "\n%10s %6s %5s %s\n\n", "COUNTRY","AREA","POP","CONTINENT"}
{ printf "%10s %6d %5d %s\n",$1,$2,$3,$4
area=area+$2; popu=popu+$3}
END { printf "\n%10s %6d %5d\n","TOTAL",area,popu }
$
```

```
$awk -f a5 country
COUNTRY AREA POP CONTINENT

USSR 8650 262 Asia
Canada 3852 24 North_America
China 3692 866 Asia
USA 3615 219 North_America
Brazil 3286 116 South_America
Australia 2968 14 Australia
India 1269 637 Asia
Argentina 1072 26 South_America
Sudan 968 19 Africa
Algeria 920 18 Africa
```

```
TOTAL 30292 2201
$
```

To print no. of lines, no. of words & no. of character in a file

```
$cat a6
BEGIN {w=0;c=0}
{w=w+NF
c=c+length($0)+1} # 1 is added for new line characer
END {printf " %6d %6d %6d %s\n",NR,w,c,ARGV[1]}
$
```

NF is a Built-in awk variable, it returns no. of fields in each record, **NR** in END pattern gives last record number, length function returns no. of characters in a string and ARGV[1] returns first argument in command line

```
$awk -f a6 country
10 40 395 country
$

$wc country
10 40 395 country
$
```

To change output field separator (":" instead of blank)

```
$cat a7
BEGIN {OFS=":"}
{print $1,$2,$3,$4}
$

$awk -f a7 country > country1

$cat country1
USSR:8650:262:Asia
Canada:3852:24:North_America
China:3692:866:Asia
USA:3615:219:North_America
Brazil:3286:116:South_America
Australia:2968:14:Australia
India:1269:637:Asia
Argentina:1072:26:South_America
Sudan:968:19:Africa
Algeria:920:18:Africa
$
```

To calculate and print population in "*Asia*", "*Africa*" and other continents

```
$cat a8
BEGIN {FS=":"}
{ if($4=="Asia")
pop[1]+=$3
else if($4=="Africa")
pop[2]+=$3
else
pop[3]+=$3 }
END { print "Asian population in million is",pop[1]
print "Africal population in million is",pop[2]
print "Others population in million is",pop[3] }
$
```

Here, FS indicates input field separator and pop is an array

```
$awk -f a8 country1
Asian population in million is 1765
Africal population in million is 37
Others population in million is 399
$
```

To calculate and print population in "*Asia*", "*Africa*" and other continents (print output with *for* loop)

```
$cat a9
BEGIN {FS=":"; coun[1]="Asian ";coun[2]="African";coun[3]="Others "}
{ if($4=="Asia")
pop[1]+=$3
else if($4=="Africa")
pop[2]+=$3
else
pop[3]+=$3 }
END { for(i=1;i<=3;i++)
printf "%s %s %d\n",coun[i],"population in million is",pop[i]
}
$
```

```
$awk -f a9 country1
Asian population in million is 1765
African population in million is 37
Others population in million is 399
$
```

To print records replace 4th field with "*America*" if 4th field contains "*America*"

1st method

```
$cat a10
BEGIN {FS=OFS=":"}
{ if(substr($4,7)=="America")
print $1,$2,$3,"America"
else
print $0 }
$
```

```
$awk -f a10 country1
USSR:8650:262:Asia
Canada:3852:24:America
China:3692:866:Asia
USA:3615:219:America
Brazil:3286:116:America
Australia:2968:14:Australia
India:1269:637:Asia
Argentina:1072:26:America
Sudan:968:19:Africa
Algeria:920:18:Africa
$
```

2nd method

```
$cat a11
BEGIN {FS=OFS=":"}
{ s = (substr($4,7)=="America") ?
sprintf("%s:%s:%s:%s",$1,$2,$3,"America"):$0

print s
}
$

$awk -f a11 country1
USSR:8650:262:Asia
Canada:3852:24:America
China:3692:866:Asia
USA:3615:219:America
Brazil:3286:116:America
Australia:2968:14:Australia
India:1269:637:Asia
Argentina:1072:26:America
Sudan:968:19:Africa
Algeria:920:18:Africa
$
```

To calculate horizontal and vertical total of a numeric file

```
$cat a12
{ i=1 ; hsum=0
while (i<=NF)
{
vsum[i]+=$i
hsum+=$i
printf "%7d ",$i
i++
}
printf "%7d\n",hsum
}
```

```
END { print ; for(j=1;j<i;j++) printf "%7d ",vsum[j] ; print }
$

$cat num
100 200 300
200 300 100
300 100 200
$

$awk -f a12 num

100 200 300 600
200 300 100 600
300 100 200 600

600 600 600
$
```

To declare a function called fact(n) to find out factorial of *n* and then calculate factorial for a list number supplied from a file

```
$cat a13
function fact(n)
{
if(n<=1)
return 1
else
return n*fact(n-1)
}

{print $1 "! is " fact($1)}
$

$cat number
1
2
3
4
5
6
7
$

$awk -f a13 number
1! is 1
2! is 2
3! is 6
4! is 24
5! is 120
6! is 720
7! is 5040
```

$

Maintenance of Data file Using Bourne shell And Awk

With the help of shell programming user can create a data file, say "*item*" which contains *6 fields* and they are separated by a colon (:). The structure of the file shown below

Item Number
Item Description
Reorder Level
Quantity on Hand
Unit of the Item
Unit price in rupees

All the procedures (**Add, Modify & Listing**) can invoke through a shell procedure called *"menu"*.

```
$sh menu


Main Menu
~~~~~~~~~

<1> Add Item

<2> Modify Item

<3> List Items

<0> Quit

Enter Choice:
```

*If user's choice is **1** then following screen will appear*

```
 Add Items
 ~~~~~~~~~

Item No :

Item Description:

Quantity on Hand:

Reorder Level :

Unit :
```

```
Unit Price(Rs.) :


Press Enter at first field to quit
```

*A sample of "**item**" data file*

```
$cat item
i001:Rice:40:25:kg:12
i003:fan:20:5::600
i002:Table:6:7::350
i004:chair:15:10::150
i005:cover file:40:25::25
$
```

*If user's choice is **2** then following screen will appear*

```
 Modify Items
 ~~~~~~~~~~~~


Item No :

Old Item Description:
New Item Description:

Old Quantity on Hand:
New Quantity on Hand:

Old Reorder Level :
New Reorder Level :

Old Unit :
New Unit :

Old Unit Price(Rs.) :
New Unit Price(Rs.) :


Press Enter at first field to quit
```

The old values of the different fields will be shown as well as prompt for new values.

*If user's choice is **3** then it will display formatted items*

```
 ITEM DETAILS Page: 1


Item no Item Description QOH Reorder_Level Unit Unit_Price
-------------------------------------------------------------------
i001 Rice 40 25 kg 12
i003 fan 20 5 600
i002 Table 6 7 350
i004 chair 15 10 150
i005 cover file 40 25 25
-------------------------------------------------------------------
:(EOF)
```

*Listing of "**menu**" file*

```
********************* MENU FOR ITEM MAINTENANCE *****************
fl='n'
ch=0
while true
do
 tput clear

 if test $fl = 'y' # If previous choice is invalid
 then
 tput cup 24 1
 echo "Invalid Choice\07\c" # \07 to echo a bell
 fi

 tput cup 0 0
 # Display a menu
 cat << M


                              Main Menu
                              ~~~~~~~~~

                   <1> Add Item

                   <2> Modify Item

                   <3> List Items

                   <0> Quit

                   Enter Choice:
                   M

 tput cup 16 41
 echo "$ch" # Display previous choice
 tput cup 16 41
```

```
 read ch # Read the choice

 case $ch in # Invoke other shell procedure
 1) sh additem;; # depending on the value of $ch
 2) sh moditem;;
 3) sh listitem;;
 0) break;;
 *) fl='y';;
 esac

 if test $ch = 1 -o $ch = 2 # If choice is 1 or 2 then
 then # system waits for users
 tput cup 24 40 # response
 echo "Press any key to continue.....\c"
 tput cup 24 70
 read wt
 fi

done
tput clear
```

*Listing of "**additem**" file*

```
#************************** ADD ITEM ***********************

tput rmso ## to turn off reverse video mode
rec=0
while true
do
 tput clear
 cat << EOT



  Add Items
  ~~~~~~~~~


 Item No :

 Item Description:

 Quantity on Hand:

 Reorder Level :

 Unit :

 Unit Price(Rs.) :
EOT
```

```
 tput smso ## to turn on reverse video mode

 tput cup 24 40
 echo "Press Enter at first field to quit\c"

 tput cup 7 34
 read itno

 if test -z "$itno"
 then
 tput rmso ## to turn off reverse video mode
 break
 fi

 tput cup 9 34
 read itdes
 tput cup 11 34
 read qoh
 tput cup 13 34
 read rlvl
 tput cup 15 34
 read unit
 tput cup 17 34
 read price


 if test -f item
 then
 if grep "^$itno:" item > /dev/null
 then
 tput cup 22
 echo "Item# $itno already exist!!"
 tput cup 24
 echo "Press any key to continue....\c"
 read wt
 tput rmso ## to turn off reverse video mode
 continue
 fi
 fi
 echo "$itno:$itdes:$qoh:$rlvl:$unit:$price">>item
 rec=`expr $rec + 1`

 tput rmso ## to turn off reverse video mode
done
tput clear
echo "$rec records added"
```

*Listing of "**moditem**" file*

```
#**************************** MODIFY ITEM ***********************

tput rmso ## to turn off reverse video mode
```

```
rec=0
while true
do
 tput clear
 cat << EOT
  Modify Items
  ~~~~~~~~~~~~~
 Item No :

 Old Item Description:
 New Item Description:

 Old Quantity on Hand:
 New Quantity on Hand:

 Old Reorder Level :
 New Reorder Level :

 Old Unit :
 New Unit :

 Old Unit Price(Rs.) :
 New Unit Price(Rs.) :
EOT
 tput smso ## to turn on reverse video mode

 tput cup 24 40
 echo "Press Enter at first field to quit\c"

 tput cup 5 38
 read itno

 if test -z "$itno"
 then
 tput rmso ## to turn off reverse video mode
 break
 fi

 if test -f item
 then
     it=`grep "^$itno:" item` ## store the searched record
     if test -z "$it"
     then
             tput cup 22
             echo "Item# $itno does not exist!!"
             tput cup 23
             echo "Press any key to continue....\c"
             read wt
             tput rmso       ## to turn off reverse video mode
             continue
     fi
 fi
```

```
## store fields value from searched record
        itdes=`echo "$it" | cut -d: -f2`
        qoh=`echo "$it" | cut -d: -f3`
        rlvl=`echo "$it" | cut -d: -f4`
        unit=`echo "$it" | cut -d: -f5`
        price=`echo "$it" | cut -d: -f6`

## display the old value
        tput cup 7 38
        echo "$itdes"
        tput cup 10 38
        echo "$qoh"
        tput cup 13 38
        echo "$rlvl"
        tput cup 16 38
        echo "$unit"
        tput cup 19 38
        echo "$price"


## read new values
        tput cup 8 38
        read itdes
        tput cup 11 38
        read qoh
        tput cup 14 38
        read rlvl
        tput cup 17 38
        read unit
        tput cup 20 38
        read price

## replace searched record with modified values
        sed "s/$it/$itno:$itdes:$qoh:$rlvl:$unit:$price/" item >item1
        mv item1 item

        rec=`expr $rec + 1` ## counts no. of records modified

        tput rmso ## to turn off reverse video mode
done
tput clear
echo "$rec records modified"
```

*Listing of "**listitem**" file*

```
#*************************** LIST ITEMS ***********************

tput rmso ## to turn off reverse video mode awk '
BEGIN{ FS=":";page=1;line=20 }
```

```
{
        if(line>19)
        {
                if(page!=1) # Page footer
                {
                        for(i=1;i<80;i++)printf "-"
                        printf "\n"
                        system("tput cup 24 0")
                        printf ":"
                        system("tput cup 24 1")
                        system("read wt")
                }
                system("tput clear") # Page heading
                system("tput cup 0")
                printf "%35sITEM DETAILS%23sPage: %3d\n\n\n"," ", " ",page
                printf "Item no%4sItem Description%10sQOH%5s Reorder_Level Unit
Unit_Price\n"," "," ",""
                        for(i=1;i<80;i++)printf "-"
                        printf "\n"
                        page+=1
                        line=5
                }



                line+=1 # Detail line
                printf "%-10s %-25s %3d%8s%5d%6s%7s %12d\n",$1,$2,$3," ",$4," ",$5,$6
}
END{
                for(i=1;i<80;i++)printf "-" # End of listing
                printf "\n"
                system("tput cup 24 0")
                printf ":(EOF)"
                system("tput cup 24 6")
                system("read wt")
                system("tput clear")
}
        ' item
```

SED Commands

Editing files specifying instructions

```
$ cat > faculty
Ranjan Dasgupta, HOD, CSE
Rajeev Chatterjee, Lect, CSE
D. Ray, HOD, EE
S. Ray, HOD, CE
Samiran Mandal, ASTT. PROF, ME
```

Using the sample input file, faculty, the following example uses the **s** command for substitution to replace “HOD” with “HEAD”

```
$ sed  ‘s/HOD/HEAD/’ faculty
Ranjan Dasgupta, HEAD, CSE
Rajeev Chatterjee, Lect, CSE
D. Ray, HEAD, EE
S. Ray, HEAD, CE
Samiran Mandal, ASTT. PROF, ME
```

Editing files with multiple instructions

Instructions can be separated specifying ‘;’

```
$ sed ‘s/HOD/HEAD/; s/CSE/Computer Science/’ faculty
Ranjan Dasgupta, HEAD, Computer Science
Rajeev Chatterjee, Lect, Computer Science
D. Ray, HEAD, EE
S. Ray, HEAd, CE
Samiran Mandal, ASTT. PROF, ME
```

Editing files with –e option

The **–e** option is necessary only when you supply more then one instruction on the command line. It tells sed to interpret the next argument as an instruction.

```
$ sed –e ‘s/HOD/HEAD/’ –e ‘s/CSE/Computer Science
Ranjan Dasgupta, HEAD, Computer Science
```

```
Rajeev Chatterjee, Lect, Computer Science
D. Ray, HEAD, EE
S. Ray, HEAD, CE
Samiran Mandal, ASTT. PROF, ME
```

Editing files with multi line entry

Use the multi line entry capabilities of the Bourne shell, press RETURN after entering a single quote and a secondary prompt(>) will be displayed for multi line input.

```
$ sed ‘
 >  s/HOD/HEAD/
>   s/CSE/Computer Science/
>   s/Lect/Lecturer/
>   s/EE/ Electrical Engineering/
>   s/ME/ Mechanical Engineering/
>  s/CE/ Continuing Education/
> s/ASTT. PROF./Assistant Professor,/ faculty

Ranjan Dasgupta, HEAD, Computer Science
Rajeev Chatterjee, Lecturer, Computer Science
D. Ray, HEAD, Electrical Engineering
S. Ray, HEAD,  Continuing Education
Samiran Mandal, Assistant Professor, Mechanical Engineering
```

Editing files with script file

It is not practical to enter longer editing scripts on the command line. That is why it is usually best to create a script file that contains the editing instructions. The editing script is simply a list of sed commands that are executed in the order in which they appear. The form, using ***–f*** option , requires that you specify the name of the script file on the command line.

```
$ cat > change
s/HOD/HEAD/
s/Lect/Lecturer/
s/CSE/Computer Science/
s/EE/Electrical Engineering/
s/ME/Mechanical Engineering/
s/CE/ Continuing Education/
s/ ASTT. PROF. / Assistant Professor,/

$ sed –f change faculty
Ranjan Dasgupta, HEAD, Computer Science
Rajeev Chatterjee, Lecturer, Computer Science
D. Ray, HEAD, Electrical Engineering
S. Ray, HEAD,  Continuing Education
Samiran Mandal, Assistant Professor,Mechanical Engineering
```

Saving the output in a file

```
$ sed –f change faculty > newfaculty
$ cat newfaculty
Ranjan Dasgupta, HEAD, Computer Science
Rajeev Chatterjee, Lecturer, Computer Science
```

```
D. Ray, HEAD, Electrical Engineering
S. Ray, HEAD,  Continuing Education
Samiran Mandal, Assistant Professor,Mechanical Engineering
```

Depressing automatic display of input lines

The default operation of sed is to output every input line. The **–n** option suppresses the automatic ouput. When specifying this option, each instruction intended to produce output must obtain a print command, **p**.

```
$ sed –n –e ‘s/HOD/HEAD/p’ faculty
Ranjan Dasgupta, HEAD, Computer Science
D. Ray, HEAD, Electrical Engineering
S. Ray, HEAD,  Continuing Education
```